Module 2.2

Submission date: 15 September 2024

Q1. WAP to demonstrate the functionality of all the methods of String class.

1. int length()
2. char charAt(int index)
3. int indexOf(int ch)
4. int indexOf(int ch, int fromIndex)
5. int indexOf(String substring)
6. int indexOf(String substring, int fromIndex)
7. lastIndexOf( )
8. String substring(int beginIndex)
9. String substring(int beginIndex, int endIndex)
10. boolean contains(CharSequence s)
11. String concat(String s)
12. boolean equals(Object o)
13. boolean equalsIgnoreCase(String s)
14. boolean isEmpty()
15. boolean equals(Object o)
16. boolean equalsIgnoreCase(String s)
17. String toLowerCase()
18. String toUpperCase()
19. int compareTo(String anotherString)
20. int compareToIgnoreCase(String anotherString)
21. String trim()
22. String replace (char oldChar, char newChar)
23. char[] toCharArray():
24. boolean startsWith(String s)
25. boolean endswith(String s)
26. static String join(CharSequence *delim*, CharSequence . . . *strs*)
27. byte[] getBytes()
28. public void getChars(int srcBeginIndex, int srcEndIndex, char[] destination, int dstBeginIndex)

```
public class StringMethodsDemo {
```

```
    public static void main(String[] args) {
        String str = "Hello, World!";
        // 1. int length()
        System.out.println("Length: " + str.length()); // Output: 13
        // 2. char charAt(int index)
        System.out.println("Character at index 7: " + str.charAt(7)); // Output: W
        // 3. int indexOf(int ch)
        System.out.println("Index of 'W': " + str.indexOf('W')); // Output: 7
        // 4. int indexOf(int ch, int fromIndex)
        System.out.println("Index of 'o' from index 5: " + str.indexOf('o', 5)); // Output: 8
        // 5. int indexOf(String substring)
        System.out.println("Index of 'World': " + str.indexOf("World")); // Output: 7
        // 6. int indexOf(String substring, int fromIndex)
        System.out.println("Index of 'o' from index 5: " + str.indexOf("o", 5)); // Output: 8
        // 7. int lastIndexOf(String substring)
        System.out.println("Last index of 'o': " + str.lastIndexOf('o')); // Output: 8
        // 8. String substring(int beginIndex)
        System.out.println("Substring from index 7: " + str.substring(7)); // Output: World!
        // 9. String substring(int beginIndex, int endIndex)
        System.out.println("Substring from index 7 to 12: " + str.substring(7, 12)); // Output: World
        // 10. boolean contains(CharSequence s)
        System.out.println("Contains 'World': " + str.contains("World")); // Output: true
        // 11. String concat(String s)
        System.out.println("Concatenation with ' How are you?': " + str.concat(" How are you?")); // Output: Hello, World!
How are you?
        // 12. boolean equals(Object o)
        System.out.println("Equals 'Hello, World!': " + str.equals("Hello, World!")); // Output: true
        // 13. boolean equalsIgnoreCase(String s)
        System.out.println("EqualsIgnoreCase 'HELLO, WORLD!': " + str.equalsIgnoreCase("HELLO, WORLD!")); //
Output: true
        // 14. boolean isEmpty()
        System.out.println("Is empty: " + str.isEmpty()); // Output: false
        // 15. boolean equals(Object o) [Repeated]
        // Already demonstrated as #12
        // 16. boolean equalsIgnoreCase(String s) [Repeated]
        // Already demonstrated as #13
        // 17. String toLowerCase()
        System.out.println("Lower case: " + str.toLowerCase()); // Output: hello, world!
        // 18. String toUpperCase()
        System.out.println("Upper case: " + str.toUpperCase()); // Output: HELLO, WORLD!
        // 19. int compareTo(String anotherString)
        System.out.println("Compare to 'Hello, World!': " + str.compareTo("Hello, World!")); // Output: 0
        // 20. int compareToIgnoreCase(String anotherString)
        System.out.println("CompareToIgnoreCase 'HELLO, WORLD!': " + str.compareToIgnoreCase("HELLO,
WORLD!")); // Output: 0
        // 21. String trim()
        String strWithSpaces = "  Hello, World!  ";
        System.out.println("Trimmed: '" + strWithSpaces.trim() + "'"); // Output: 'Hello, World!'
        // 22. String replace(char oldChar, char newChar)
        System.out.println("Replace 'o' with '0': " + str.replace('o', '0')); // Output: Hell0, W0rld!
        // 23. char[] toCharArray()
        char[] chars = str.toCharArray();
        System.out.print("To char array: ");
```

```
        for (char c : chars) {
            System.out.print(c + " "); // Output: H e l l o ,   W o r l d !
        }
        System.out.println();
        // 24. boolean startsWith(String s)
        System.out.println("Starts with 'Hello': " + str.startsWith("Hello")); // Output: true
        // 25. boolean endsWith(String s)
        System.out.println("Ends with 'World!': " + str.endsWith("World!")); // Output: true
        // 26. static String join(CharSequence delim, CharSequence... strs)
        System.out.println("Join 'A', 'B', 'C' with ', ': " + String.join(", ", "A", "B", "C")); // Output: A, B, C
        // 27. byte[] getBytes()
        byte[] bytes = str.getBytes();
        System.out.print("Bytes: ");
        for (byte b : bytes) {
            System.out.print(b + " "); // Output: bytes corresponding to the characters in the string
        }
        System.out.println();
        // 28. public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
        char[] destination = new char[5];
        str.getChars(7, 12, destination, 0);
        System.out.print("Get chars (7 to 12): ");
        for (char c : destination) {
            System.out.print(c); // Output: World
        }
        System.out.println();
    }
}
```

```
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasi
ngh/Library/Application\ Support/Code/User/globalStorage/pleiades.java-extensi
on-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExcept
ionMessages -cp /Users/shikhasingh/Library/Application\ Support/Code/User/work
spaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\ 2
.2_e2a78372/bin StringMethodsDemo
Length: 13
Character at index 7: W
Index of 'W': 7
Index of 'o' from index 5: 8
Index of 'World': 7
Index of 'o' from index 5: 8
Last index of 'o': 8
Substring from index 7: World!
Substring from index 7 to 12: World
Contains 'World': true
Concatenation with ' How are you?': Hello, World! How are you?
Equals 'Hello, World!': true
EqualsIgnoreCase 'HELLO, WORLD!': true
Is empty: false
Lower case: hello, world!
Upper case: HELLO, WORLD!
Compare to 'Hello, World!': 0
CompareToIgnoreCase 'HELLO, WORLD!': 0
Trimmed: 'Hello, World!'
Replace 'o' with '0': Hell0, W0rld!
To char array: H e l l o ,   W o r l d !
Starts with 'Hello': true
Ends with 'World!': true
Join 'A', 'B', 'C' with ', ': A, B, C
Bytes: 72 101 108 108 111 44 32 87 111 114 108 100 33
Get chars (7 to 12): World
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q2. WAP to demonstrate the functionality of all the methods of

StringBuilder class.

1. append() method for all data types
2. insert() method for all data types
3. StringBuilder insert(int offset, char[] str)
4. StringBuilder insert(int index, char[] str, int offset, int len)
5. StringBuilder insert(int dstOffset, CharSequence s)
6. StringBuilder insert(int dstOffset, CharSequence s, int start, int end)
7. StringBuilder insert(int offset, Object obj)
8. StringBuilder replace(int startIndex, int endIndex, String str)
9. StringBuilder delete(int startIndex, int endIndex)
10. public StringBuilder reverse()
11. public int capacity()
12. public void ensureCapacity(int minimumCapacity)
13. public char charAt(int index)
14. public int length()
15. public String substring(int beginIndex)
16. public String substring(int beginIndex, int endIndex)
17. int compareTo(StringBuilder another)
18. void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
19. StringBuilderdeleteCharAt(int index)
20. int lastIndexOf(String str)
21. int lastIndexOf(String str, int fromIndex)
22. int indexOf(String str)
23. int indexOf(String str, int fromIndex)
24. void setCharAt(int index, char ch)
25. String toString()

```
public class StringBuilderMethodsDemo {
    public static void main(String[] args) {
        // Creating a StringBuilder object for demonstration
        StringBuilder sb = new StringBuilder("Hello");
        // 1. append() method for all data types
        sb.append(" World"); // Appending a String
        sb.append(123); // Appending an integer
        sb.append(45.67); // Appending a double
        sb.append(true); // Appending a boolean
        sb.append('!'); // Appending a char
        System.out.println("After append(): " + sb);
        // 2. insert() method for all data types
```

```
sb.insert(5, " Beautiful"); // Inserting a String
sb.insert(15, 789); // Inserting an integer
sb.insert(18, 56.78); // Inserting a double
sb.insert(24, false); // Inserting a boolean
sb.insert(30, 'X'); // Inserting a char
System.out.println("After insert(): " + sb);
// 3. insert(int offset, char[] str)
char[] chars = {' ', 'G', 'o', 'o', 'd', ' '};
sb.insert(32, chars); // Inserting char array
System.out.println("After insert(char[]): " + sb);
// 4. insert(int index, char[] str, int offset, int len)
char[] moreChars = {' ', 'D', 'a', 'y'};
sb.insert(33, moreChars, 1, 3); // Inserting part of char array
System.out.println("After insert(char[], offset, len): " + sb);
// 5. insert(int dstOffset, CharSequence s)
sb.insert(36, "Everyone"); // Inserting CharSequence
System.out.println("After insert(CharSequence): " + sb);
// 6. insert(int dstOffset, CharSequence s, int start, int end)
sb.insert(45, "Nice", 0, 3); // Inserting part of CharSequence
System.out.println("After insert(CharSequence, start, end): " + sb);
// 7. insert(int offset, Object obj)
sb.insert(48, new Object()); // Inserting Object
System.out.println("After insert(Object): " + sb);
// 8. replace(int startIndex, int endIndex, String str)
sb.replace(48, 55, "Object"); // Replacing a substring
System.out.println("After replace(): " + sb);
// 9. delete(int startIndex, int endIndex)
sb.delete(48, 55); // Deleting a substring
System.out.println("After delete(): " + sb);
// 10. reverse()
sb.reverse(); // Reversing the entire string
System.out.println("After reverse(): " + sb);
// 11. capacity()
System.out.println("Capacity: " + sb.capacity()); // Output current capacity
// 12. ensureCapacity(int minimumCapacity)
sb.ensureCapacity(100); // Ensuring minimum capacity
System.out.println("Capacity after ensureCapacity(100): " + sb.capacity());
// 13. charAt(int index)
System.out.println("Character at index 0: " + sb.charAt(0)); // Output character at index 0
// 14. length()
System.out.println("Length: " + sb.length()); // Output the length of the string
// 15. substring(int beginIndex)
System.out.println("Substring from index 5: " + sb.substring(5)); // Extracts substring from index 5
// 16. substring(int beginIndex, int endIndex)
System.out.println("Substring from index 5 to 10: " + sb.substring(5, 10)); // Extracts substring from index 5 to 10
// 17. compareTo(StringBuilder another)
StringBuilder sb2 = new StringBuilder("AnotherString");
System.out.println("CompareTo sb2: " + sb.compareTo(sb2)); // Compares this StringBuilder with another
// 18. getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
char[] dest = new char[5];
sb.getChars(5, 10, dest, 0); // Copies characters to the char array
System.out.print("Get chars (5 to 10): ");
for (char c : dest) {
```

```
            System.out.print(c); // Output copied characters
        }
        System.out.println();
        // 19. deleteCharAt(int index)
        sb.deleteCharAt(0); // Deleting character at a specific index
        System.out.println("After deleteCharAt(0): " + sb);
        // 20. lastIndexOf(String str)
        System.out.println("Last index of 'a': " + sb.lastIndexOf("a")); // Finds the last occurrence of a substring
        // 21. lastIndexOf(String str, int fromIndex)
        System.out.println("Last index of 'a' from index 10: " + sb.lastIndexOf("a", 10)); // Finds the last occurrence of a
substring from a specific index
        // 22. indexOf(String str)
        System.out.println("Index of 'World': " + sb.indexOf("World")); // Finds the first occurrence of a substring
        // 23. indexOf(String str, int fromIndex)
        System.out.println("Index of 'World' from index 10: " + sb.indexOf("World", 10)); // Finds the first occurrence of a
substring from a specific index
        // 24. setCharAt(int index, char ch)
        sb.setCharAt(1, 'Z'); // Setting character at a specific index
        System.out.println("After setCharAt(1, 'Z'): " + sb);
        // 25. toString()
        String str = sb.toString(); // Converting StringBuilder to String
        System.out.println("To String: " + str);
    }
}
```

```
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasingh/Library/Application\ Support/Code/User/globalStorage/pleia
des.java-extension-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/shikhasingh/Library/App
lication\ Support/Code/User/workspaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\ 2.2_e2a78372/bin StringBuilderM
ethodsDemo
After append(): Hello World12345.67true!
After insert(): Hello Beautiful78956.78 falseWXorld12345.67true!
After insert(char[]): Hello Beautiful78956.78 falseWXo Good rld12345.67true!
After insert(char[], offset, len): Hello Beautiful78956.78 falseWXo DayGood rld12345.67true!
After insert(CharSequence): Hello Beautiful78956.78 falseWXo DayEveryoneGood rld12345.67true!
After insert(CharSequence, start, end): Hello Beautiful78956.78 falseWXo DayEveryoneGNicood rld12345.67true!
After insert(Object): Hello Beautiful78956.78 falseWXo DayEveryoneGNicjava.lang.Object@68f7aae2ood rld12345.67true!
After replace(): Hello Beautiful78956.78 falseWXo DayEveryoneGNicObjectng.Object@68f7aae2ood rld12345.67true!
After delete(): Hello Beautiful78956.78 falseWXo DayEveryoneGNicg.Object@68f7aae2ood rld12345.67true!
After reverse(): !eurt76.54321dlr doo2eaa7f86@tcejbO.gciNGenoyrevEyaD oXWeslaf 87.65987lufituaeB olleH
Capacity: 182
Capacity after ensureCapacity(100): 182
Character at index 0: !
Length: 85
Substring from index 5: 76.54321dlr doo2eaa7f86@tcejbO.gciNGenoyrevEyaD oXWeslaf 87.65987lufituaeB olleH
Substring from index 5 to 10: 76.54
CompareTo sb2: -32
Get chars (5 to 10): 76.54
After deleteCharAt(0): eurt76.54321dlr doo2eaa7f86@tcejbO.gciNGenoyrevEyaD oXWeslaf 87.65987lufituaeB olleH
Last index of 'a': 75
Last index of 'a' from index 10: -1
Index of 'World': -1
Index of 'World' from index 10: -1
After setCharAt(1, 'Z'): eZrt76.54321dlr doo2eaa7f86@tcejbO.gciNGenoyrevEyaD oXWeslaf 87.65987lufituaeB olleH
To String: eZrt76.54321dlr doo2eaa7f86@tcejbO.gciNGenoyrevEyaD oXWeslaf 87.65987lufituaeB olleH
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q3. WAP to demonstrate the functionality of the following methods of StringBuffer class.

1. intcapacity()
2. char charAt(int index)]
3. StringBuffer delete(int start, int end)
4. StringBuffer deleteCharAt(int index)
5. void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

6. int indexOf(String str)
7. int indexOf(String str, int fromIndex)
8. int lastIndexOf(String str)
9. int lastIndexOf(String str, int fromIndex)
10. int length()
11. StringBuffer replace(int start, int end, String str)
12. StringBuffer reverse()
13. void setCharAt(int index, char ch)
14. StringBuffer replace(int start, int end, String str)
15. StringBuffer reverse()
16. void setCharAt(int index, char ch)
17. String substring(int start)
18. String substring(int start, int end)
19. String toString()

```
public class StringBufferMethodsDemo {
    public static void main(String[] args) {
        // Creating a StringBuffer object for demonstration
        StringBuffer sb = new StringBuffer("Hello, World!");
        // 1. int capacity()
        System.out.println("Initial capacity: " + sb.capacity()); // Output the current capacity
        // 2. char charAt(int index)
        System.out.println("Character at index 7: " + sb.charAt(7)); // Output character at index 7
        // 3. StringBuffer delete(int start, int end)
        sb.delete(5, 12); // Delete characters from index 5 to 11
        System.out.println("After delete(5, 12): " + sb);
        // 4. StringBuffer deleteCharAt(int index)
        sb.deleteCharAt(5); // Delete character at index 5
        System.out.println("After deleteCharAt(5): " + sb);
        // 5. void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
        char[] dest = new char[5];
        sb.getChars(0, 5, dest, 0); // Copy characters to the char array
        System.out.print("Get chars (0 to 5): ");
        for (char c : dest) {
            System.out.print(c); // Output copied characters
        }
        System.out.println();
        // 6. int indexOf(String str)
        System.out.println("Index of 'World': " + sb.indexOf("World")); // Find the first occurrence of substring
        // 7. int indexOf(String str, int fromIndex)
        System.out.println("Index of 'World' from index 0: " + sb.indexOf("World", 0)); // Find the first occurrence of
substring from index
        // 8. int lastIndexOf(String str)
        System.out.println("Last index of 'World': " + sb.lastIndexOf("World")); // Find the last occurrence of substring
        // 9. int lastIndexOf(String str, int fromIndex)
        System.out.println("Last index of 'World' from index 5: " + sb.lastIndexOf("World", 5)); // Find the last occurrence
of substring from index
```

```
        // 10. int length()
        System.out.println("Length: " + sb.length()); // Output the length of the StringBuffer
        // 11. StringBuffer replace(int start, int end, String str)
        sb.replace(5, 10, "Universe"); // Replace characters from index 5 to 9 with the specified string
        System.out.println("After replace(5, 10, 'Universe'): " + sb);
        // 12. StringBuffer reverse()
        sb.reverse(); // Reverse the entire sequence
        System.out.println("After reverse(): " + sb);
        // 13. void setCharAt(int index, char ch)
        sb.setCharAt(0, 'H'); // Set character at index 0
        System.out.println("After setCharAt(0, 'H'): " + sb);
        // 14. StringBuffer replace(int start, int end, String str) [Repeated]
        // Already demonstrated as #11
        // 15. StringBuffer reverse() [Repeated]
        // Already demonstrated as #12
        // 16. void setCharAt(int index, char ch) [Repeated]
        // Already demonstrated as #13
        // 17. String substring(int start)
        System.out.println("Substring from index 0: " + sb.substring(0)); // Extract substring from index 0
        // 18. String substring(int start, int end)
        System.out.println("Substring from index 0 to 5: " + sb.substring(0, 5)); // Extract substring from index 0 to 5
        // 19. String toString()
        String str = sb.toString(); // Convert StringBuffer to String
        System.out.println("To String: " + str);
    }
}
```

```
/Users/shikhasingh/.zshrc:1: command not found: his
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasingh/Library/Application\ Support/Code/User/globalStorage/pleiades.java-extension-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/shikhasingh/Library/Application\ Support/Code/User/workspaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\ 2.2_e2a78372/bin StringBufferMethodsDemo
Initial capacity: 29
Character at index 7: W
After delete(5, 12): Hello!
After deleteCharAt(5): Hello
Get chars (0 to 5): Hello
Index of 'World': -1
Index of 'World' from index 0: -1
Last index of 'World': -1
Last index of 'World' from index 5: -1
Length: 5
After replace(5, 10, 'Universe'): HelloUniverse
After reverse(): esrevinUolleH
After setCharAt(0, 'H'): HsrevinUolleH
Substring from index 0: HsrevinUolleH
Substring from index 0 to 5: Hsrev
To String: HsrevinUolleH
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q4. WAP to demonstrate the how to handle ArrayIndexOutOfBoundException, StringIndexOutOfBoundException in a program using multiple catch clause.

```
public class ExceptionHandlingDemo {
    public static void main(String[] args) {
        // Example data
        int[] array = {1, 2, 3, 4, 5};
        String str = "Hello";
        try {
            // Code that might throw exceptions
```

```
        // ArrayIndexOutOfBoundsException
        System.out.println("Accessing array element at index 10: " + array[10]);
        // StringIndexOutOfBoundsException
        System.out.println("Accessing string character at index 20: " + str.charAt(20));

    } catch (ArrayIndexOutOfBoundsException e) {
        // Handle ArrayIndexOutOfBoundsException
        System.out.println("Exception caught: ArrayIndexOutOfBoundsException");
        System.out.println("Details: " + e.getMessage());
        e.printStackTrace();

    } catch (StringIndexOutOfBoundsException e) {
        // Handle StringIndexOutOfBoundsException
        System.out.println("Exception caught: StringIndexOutOfBoundsException");
        System.out.println("Details: " + e.getMessage());
        e.printStackTrace();

    } finally {
        // This block will always execute, regardless of whether an exception was thrown
        System.out.println("Finally block executed.");
    }
  }
}
```

```
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasingh/Library/Application\ Support/Code/User/global
des.java-extension-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/shikhasing
lication\ Support/Code/User/workspaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\ 2.2_e2a78372/bin E
ingDemo
Exception caught: ArrayIndexOutOfBoundsException
Details: Index 10 out of bounds for length 5
java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
        at ExceptionHandlingDemo.main(ExceptionHandlingDemo.java:11)
Finally block executed.
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q5.WAP to demonstrate nested try catch statements.

```
public class NestedTryCatchDemo {
  public static void main(String[] args) {
    try {
      // Outer try block
      System.out.println("Outer try block");
      // Nested try-catch block
      try {
        System.out.println("Inner try block");
        // This will throw ArithmeticException
        int result = 10 / 0;

      } catch (ArithmeticException e) {
        // Handling ArithmeticException in the inner try-catch
        System.out.println("Caught ArithmeticException in inner try-catch");
        System.out.println("Details: " + e.getMessage());
        e.printStackTrace();
      }

      // This will throw ArrayIndexOutOfBoundsException
```

```
        int[] array = new int[5];
        System.out.println("Accessing array element at index 10: " + array[10]);
    } catch (ArrayIndexOutOfBoundsException e) {
        // Handling ArrayIndexOutOfBoundsException in the outer try-catch
        System.out.println("Caught ArrayIndexOutOfBoundsException in outer try-catch");
        System.out.println("Details: " + e.getMessage());
        e.printStackTrace();
    } catch (Exception e) {
        // Handling any other exceptions
        System.out.println("Caught Exception in outer try-catch");
        System.out.println("Details: " + e.getMessage());
        e.printStackTrace();
    } finally {
        // This block will always execute
        System.out.println("Finally block executed.");
    }
  }
}
```

```
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasingh/Library/Application\ Support/Code/User/globalStorage
des.java-extension-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/shikhasingh/Libra
lication\ Support/Code/User/workspaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\ 2.2_e2a78372/bin NestedTr
Demo
Outer try block
Inner try block
Caught ArithmeticException in inner try-catch
Details: / by zero
java.lang.ArithmeticException: / by zero
        at NestedTryCatchDemo.main(NestedTryCatchDemo.java:12)
Caught ArrayIndexOutOfBoundsException in outer try-catch
Details: Index 10 out of bounds for length 5
java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
        at NestedTryCatchDemo.main(NestedTryCatchDemo.java:23)
Finally block executed.
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q6. WAP to demonstrate application of throw in exception handling.

```
// Define a custom exception class
class InvalidAgeException extends Exception {
   public InvalidAgeException(String message) {
      super(message);
   }
}
public class ThrowDemo {

   // Method to check if the age is valid
   public static void checkAge(int age) throws InvalidAgeException {
      if (age < 0 || age > 150) {
         // Throwing the custom exception if the age is invalid
         throw new InvalidAgeException("Age is not valid: " + age);
      } else {
         System.out.println("Age is valid: " + age);
      }
   }
   public static void main(String[] args) {
      try {
         // Test with valid age
         checkAge(25);
         // Test with invalid age
         checkAge(-5);
```

```
        } catch (InvalidAgeException e) {
            // Handle the custom exception
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasingh/Library/Application\ Support/Code/User/globalStorage/pleiades.java-extension-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/shikhasingh/Library/Application\ Support/Code/User/workspaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\ 2.2_e2a78372/bin ThrowDemo
Age is valid: 25
Caught exception: Age is not valid: -5
InvalidAgeException: Age is not valid: -5
        at ThrowDemo.checkAge(ThrowDemo.java:14)
        at ThrowDemo.main(ThrowDemo.java:26)
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q7. WAP to demonstrate application of throws in exception handling.

```
// Define a custom exception class
class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}
// A class demonstrating the use of 'throws'
public class ThrowsDemo {

    // Method that declares it might throw a CustomException
    public static void riskyMethod() throws CustomException {
        // Simulate some condition that might lead to an exception
        boolean errorOccurred = true;
        if (errorOccurred) {
            throw new CustomException("An error occurred in riskyMethod");
        }
        System.out.println("riskyMethod executed successfully");
    }
    public static void main(String[] args) {
        try {
            // Call the method that might throw an exception
            riskyMethod();
        } catch (CustomException e) {
            // Handle the custom exception
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
        // Another method call that could be used to show no exception scenario
        try {
            // Change the condition to false to see the successful execution
            boolean errorOccurred = false;
            if (errorOccurred) {
                riskyMethod();
            } else {
                System.out.println("No exception thrown, method executed successfully.");
            }
        } catch (CustomException e) {
            // Handle the custom exception
            System.out.println("Caught exception: " + e.getMessage());
```

```
        e.printStackTrace();
    }
  }
}
```

```
/Users/shikhasingh/.zshrc:1: command not found: His
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasingh/Library/Application\ Support/Code/User/globalStorage/pleiades.java-extension-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/shikhasingh/Library/Application\ Support/Code/User/workspaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\ 2.2_e2a78372/bin ThrowsDemo
Caught exception: An error occurred in riskyMethod
CustomException: An error occurred in riskyMethod
        at ThrowsDemo.riskyMethod(ThrowsDemo.java:16)
        at ThrowsDemo.main(ThrowsDemo.java:24)
No exception thrown, method executed successfully.
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q8. WAP to demonstrate application of try, catch, finally in exception handling. Demonstrate the sequence in which these clauses will be executed.

```
public class TryCatchFinallyDemo {
   public static void main(String[] args) {
      System.out.println("Program started");
      try {
         System.out.println("Inside try block");
         // Simulate an exception
         int result = 10 / 0; // This will throw ArithmeticException
         // This line will not be executed
         System.out.println("This line will not be executed");

      } catch (ArithmeticException e) {
         System.out.println("Caught ArithmeticException");
         System.out.println("Exception message: " + e.getMessage());
         e.printStackTrace();

      } finally {
         System.out.println("Inside finally block");
      }
      System.out.println("Program ended");
   }
}
```

```
/Users/shikhasingh/.zshrc:1: command not found: His
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasingh/Library/Application\ Support/Code/User/globalStorage/pleiades.java-extension-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/shikhasingh/Library/Application\ Support/Code/User/workspaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\ 2.2_e2a78372/bin TryCatchFinallyDemo
Program started
Inside try block
Caught ArithmeticException
Exception message: / by zero
java.lang.ArithmeticException: / by zero
        at TryCatchFinallyDemo.main(TryCatchFinallyDemo.java:9)
Inside finally block
Program ended
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q9. WAP for BankingApplicationDemo.

Create a class Account which has following:

- Instance variables:

    - int accountNo,
    - double balance
  - Methods:
    - void deposit(double amt)
    - void withdraw(double amt)

Create your own custom exception named as "InsufficientFundsException" If amt to be withdrawn is greater than balance, then throw "InsufficientFundsException" and display appropriate message.

```
// Define a custom exception class
class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}
// Define the Account class
class Account {
    private int accountNo;
    private double balance;
    // Constructor to initialize account number and balance
    public Account(int accountNo, double initialBalance) {
        this.accountNo = accountNo;
        this.balance = initialBalance;
    }
    // Method to deposit money into the account
    public void deposit(double amt) {
        if (amt > 0) {
            balance += amt;
            System.out.println("Deposited: $" + amt);
        } else {
            System.out.println("Deposit amount must be positive.");
        }
    }
    // Method to withdraw money from the account
    public void withdraw(double amt) throws InsufficientFundsException {
        if (amt > balance) {
            throw new InsufficientFundsException("Insufficient funds. Available balance: $" + balance);
        } else if (amt > 0) {
            balance -= amt;
            System.out.println("Withdrew: $" + amt);
        } else {
            System.out.println("Withdrawal amount must be positive.");
        }
    }
    // Method to get the current balance
    public double getBalance() {
        return balance;
```

```
    }
    // Method to get account number
    public int getAccountNo() {
        return accountNo;
    }
}
// Main class to test the Account and custom exception
public class BankingApplicationDemo {
    public static void main(String[] args) {
        // Create an account with an initial balance
        Account myAccount = new Account(12345, 1000.00);
        // Test deposit and withdraw methods
        try {
            System.out.println("Initial Balance: $" + myAccount.getBalance());
            // Deposit money
            myAccount.deposit(500.00);
            System.out.println("Balance after deposit: $" + myAccount.getBalance());
            // Withdraw money
            myAccount.withdraw(200.00);
            System.out.println("Balance after withdrawal: $" + myAccount.getBalance());
            // Attempt to withdraw more than the balance
            myAccount.withdraw(2000.00);

        } catch (InsufficientFundsException e) {
            // Handle the custom exception
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```

```
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasingh/Library/Application\ Support/Code/User/glo
des.java-extension-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/shikhas
lication\ Support/Code/User/workspaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\ 2.2_e2a78372/bi
tionDemo
Initial Balance: $1000.0
Deposited: $500.0
Balance after deposit: $1500.0
Withdrew: $200.0
Balance after withdrawal: $1300.0
Exception caught: Insufficient funds. Available balance: $1300.0
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q10. Explain assertion in Java. WAP to demonstrate how to use assert statement.

```
public class AssertionDemo {
    // Method to calculate the square root
    public static double calculateSquareRoot(double number) {
        // Assert that the number is non-negative
        assert number >= 0 : "Number should be non-negative. Given: " + number;
        // Calculate and return the square root
        return Math.sqrt(number);
    }
    public static void main(String[] args) {
        // Test with a positive number
        System.out.println("Square root of 16: " + calculateSquareRoot(16));
        // Test with a negative number to trigger assertion
```

```
        try {
            System.out.println("Square root of -9: " + calculateSquareRoot(-9));
        } catch (AssertionError e) {
            System.out.println("AssertionError caught: " + e.getMessage());
        }
    }
}
```

```
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %  /usr/bin/env /Users/shikhasingh/Library/Application\ Suppo
des.java-extension-pack-jdk/java/latest/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp
lication\ Support/Code/User/workspaceStorage/4aca07d22ca3f8be56d21ecfdc0916d2/redhat.java/jdt_ws/Assignment\

Square root of 16: 4.0
Square root of -9: NaN
shikhasingh@Shikhas-MacBook-Air Assignment 2.2 %
```

Q11. Differentiate between String, StringBuilder, StringBuffer.

1. String

- **Immutability**:
  - **Immutable**: Once a String object is created, it cannot be changed. Any modification to a String creates a new String object.
- **Performance**:
  - **Less Efficient for Mutations**: Due to immutability, frequent modifications (like concatenations) can lead to inefficient performance because new objects are created every time a change is made.
- **Thread-Safety**:
  - **Thread-Safe**: Because String objects are immutable, they are inherently thread-safe.
- **Usage**:
  - Best used when you need a constant and unchangeable text value, or when string values are not modified frequently.
- **Example**:
  java
  Copy code

  String str = "Hello";
- str = str + " World"; // Creates a new String object
-

2. StringBuilder

- **Mutability**:
  - **Mutable**: StringBuilder objects can be modified after they are created without creating new objects.
- **Performance**:
  - **More Efficient for Mutations**: Designed for scenarios where strings are modified frequently. It is more efficient than String when performing multiple changes to the text because it doesn't create a new object with each modification.
- **Thread-Safety**:
  - **Not Thread-Safe**: StringBuilder is not synchronized, so it is not thread-safe. If multiple threads access a StringBuilder instance concurrently, external synchronization is needed.
- **Usage**:

    - Suitable for use cases where strings need to be built or modified dynamically and are not accessed by multiple threads simultaneously.
- **Example**:
  java
  Copy code

  StringBuilder sb = new StringBuilder("Hello");
- sb.append(" World"); // Modifies the existing object
-

3. StringBuffer

- **Mutability**:
    - **Mutable**: Like StringBuilder, StringBuffer objects can be modified after creation.
- **Performance**:
    - **Less Efficient for Mutations Compared to StringBuilder**: StringBuffer is similar to StringBuilder but comes with synchronization overhead, which can impact performance.
- **Thread-Safety**:
    - **Thread-Safe**: StringBuffer is synchronized, which means it is thread-safe. It ensures that only one thread can access the StringBuffer instance at a time, making it safe for use in concurrent environments.
- **Usage**:
    - Useful in multi-threaded scenarios where string modifications are needed but thread-safety is a concern.
- **Example**:
  java
  Copy code

  StringBuffer sbf = new StringBuffer("Hello");
- sbf.append(" World"); // Modifies the existing object
-

Summary of Differences

Feature	String	StringBuilder	StringBuffer
Immutability	Immutable	Mutable	Mutable
Thread-Safety	Thread-Safe	Not Thread-Safe	Thread-Safe
Performance	Less efficient for modifications	More efficient for modifications	Less efficient due to synchronization
Use Case	Fixed text values, infrequent modifications	Frequent modifications in single-threaded scenarios	Frequent modifications in multi-threaded scenarios

Q12. Differentiate between abstract class and Interface

Abstract Class vs. Interface

Feature	Abstract Class	Interface
Definition	A class that cannot be instantiated and may contain abstract methods (methods without implementation) and concrete methods (methods with implementation).	A reference type in Java that can contain only constants, method signatures (abstract methods), default methods, static methods, and nested types.
Methods	Can contain both abstract methods (without body) and concrete methods (with body).	Can contain abstract methods (without body), default methods (with body), and static methods.
Fields	Can have instance fields with or without initialization.	Can only have public, static, final fields (constants).
Constructors	Can have constructors.	Cannot have constructors.
Access Modifiers	Methods and fields can have access modifiers (e.g., private, protected, public).	All methods in interfaces are implicitly public.
Inheritance	A class can inherit from only one abstract class (single inheritance).	A class can implement multiple interfaces (multiple inheritance).
Implementation	Subclasses of an abstract class must implement all abstract methods of the abstract class.	Classes implementing an interface must provide implementations for all abstract methods declared in the interface.
Inheritance Hierarchy	Supports single inheritance. A class can inherit from only one abstract class.	Supports multiple inheritance. A class can implement multiple interfaces.
Abstract Method Implementation	Abstract methods are meant to be overridden by subclasses.	Methods can be overridden, but interfaces provide a contract rather than implementation, except for default methods.
Default Methods	Not applicable.	Can have default methods with a default implementation.
Static Methods	Can have static methods with implementations.	Can have static methods with implementations.
Usage	Used when you need to define a base class with common methods and fields that can be shared among multiple derived classes.	Used to define a contract that multiple classes can implement, providing a way to achieve polymorphism and multiple inheritance.
Extending	A class can extend only one abstract class.	A class can implement multiple interfaces.

Example

***Abstract Class**

```
abstract class Animal {
    String name;
    // Abstract method (does not have a body)
    abstract void makeSound();
    // Regular method
    public void sleep() {
        System.out.println(name + " is sleeping.");
    }
}
class Dog extends Animal {
    Dog(String name) {
        this.name = name;
    }
```

```
    // Providing implementation of the abstract method
    @Override
    void makeSound() {
        System.out.println(name + " barks.");
    }
}
```

***Interface**

```
interface Animal {
    // Abstract method (does not have a body)
    void makeSound();
    // Default method
    default void sleep() {
        System.out.println("Animal is sleeping.");
    }
    // Static method
    static void info() {
        System.out.println("Animals are living beings.");
    }
}
class Dog implements Animal {
    private String name;
    Dog(String name) {
        this.name = name;
    }
    // Providing implementation of the abstract method
    @Override
    public void makeSound() {
        System.out.println(name + " barks.");
    }
}
```

Summary

- **Abstract Classes**:
    - Can have both abstract and concrete methods.
    - Can have fields and constructors.
    - Supports single inheritance.
    - Used to provide a common base class with shared functionality.
- **Interfaces**:
    - Can have abstract methods, default methods, and static methods.
    - Can only have constants.
    - Supports multiple inheritance.
    - Used to define a contract that implementing classes must follow, facilitating polymorphism and multiple inheritance.

Q13. Differentiate between Method overloading and method overriding.

Method Overloading vs. Method Overriding

Feature	Method Overloading	Method Overriding
Definition	Occurs when multiple methods in the same class have the same name but different parameter lists (different type or number of parameters).	Occurs when a subclass provides a specific implementation of a method that is already defined in its superclass with the same name and parameter list.
Purpose	Provides multiple ways to perform a similar task with different input parameters.	Allows a subclass to provide a specific implementation of a method that is already defined in the superclass, ensuring that the subclass's version of the method is used.
Method Signature	Different method signatures (different parameters).	Same method signature (same name and parameters) as the method in the superclass.
Return Type	Can have different return types but is not distinguished by return type alone. Return type alone does not affect method overloading.	Must have the same return type as the method in the superclass or a subtype of it (covariant return type).
Access Modifiers	Access modifiers can be different for overloaded methods (e.g., public, protected, private).	The overriding method must have the same or less restrictive access modifier than the method in the superclass.
Static Methods	Can be overloaded. Static methods are resolved at compile time.	Cannot be overridden by static methods. Static methods are bound at compile time and do not exhibit polymorphism.
Inheritance	Not required. Overloading can occur within the same class.	Required. Overriding involves a subclass and its superclass.
Polymorphism	Does not support runtime polymorphism. Overloaded methods are resolved at compile time.	Supports runtime polymorphism. The method to be executed is determined at runtime based on the object's actual type.
Example	java public void display(int a) { ... } public void display(double a) { ... }	java @Override public void display() { ... } // Overrides superclass method

Examples

—Method Overloading

```
class MathOperations {
    // Method to add two integers
    public int add(int a, int b) {
        return a + b;
    }
    // Method to add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }
    // Method to add two double numbers
    public double add(double a, double b) {
        return a + b;
    }
}
public class OverloadingDemo {
    public static void main(String[] args) {
        MathOperations math = new MathOperations();

        System.out.println("Sum of two integers: " + math.add(5, 10));
        System.out.println("Sum of three integers: " + math.add(5, 10, 15));
        System.out.println("Sum of two doubles: " + math.add(5.5, 10.5));
```

```
    }
}
```
—Method Overriding
```
class Animal {
    // Method in the superclass
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}
class Dog extends Animal {
    // Overriding the method in the subclass
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }
}
public class OverridingDemo {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myDog = new Dog();
        myAnimal.makeSound(); // Calls the method in Animal class
        myDog.makeSound();    // Calls the overridden method in Dog class
    }
}
```
Summary

- **Method Overloading**:
    - Occurs within the same class.
    - Involves methods with the same name but different parameter lists.
    - Resolved at compile time.
    - Does not support runtime polymorphism.
- **Method Overriding**:
    - Occurs in a subclass.
    - Involves methods with the same name and parameter list as in the superclass.
    - Supports runtime polymorphism.
    - The overriding method must have the same or a less restrictive access modifier than the method in the superclass.

Q14. Differentiate between Compile time polymorphism and run time polymorphism.

Compile-Time Polymorphism vs. Runtime Polymorphism

Feature	Compile-Time Polymorphism	Runtime Polymorphism
Definition	Also known as method overloading or static polymorphism. Occurs when multiple methods have the same name but different parameters within the same class.	Also known as method overriding or dynamic polymorphism. Occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.

Binding	Resolved at compile time. Method calls are bound to the method implementations during compilation.	Resolved at runtime. Method calls are bound to the method implementations at runtime based on the object's actual type.
Method Signature	Involves methods with the same name but different parameter lists (different type or number of parameters).	Involves methods with the same name and parameter list in both superclass and subclass.
Implementation	Achieved through method overloading.	Achieved through method overriding.
Polymorphism Type	Static Polymorphism.	Dynamic Polymorphism.
Performance	Generally faster as method resolution happens at compile time.	May involve a slight performance overhead due to dynamic method resolution at runtime.
Example	java public class Example { public void show(int a) { ... } public void show(double a) { ... } }	java class Animal { public void makeSound() { ... } } class Dog extends Animal { @Override public void makeSound() { ... } }

Examples

—Compile-Time Polymorphism (Method Overloading)

```
class Calculator {
    // Method to add two integers
    public int add(int a, int b) {
        return a + b;
    }
    // Method to add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }
    // Method to add two double numbers
    public double add(double a, double b) {
        return a + b;
    }
}
public class CompileTimePolymorphismDemo {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("Sum of two integers: " + calc.add(5, 10));
        System.out.println("Sum of three integers: " + calc.add(5, 10, 15));
        System.out.println("Sum of two doubles: " + calc.add(5.5, 10.5));
    }
}
```

—Runtime Polymorphism (Method Overriding)

```
class Animal {
    // Method in the superclass
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}
class Dog extends Animal {
    // Overriding the method in the subclass
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
```

```
    }
}
public class RuntimePolymorphismDemo {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myDog = new Dog(); // Upcasting
        myAnimal.makeSound(); // Calls the method in Animal class
        myDog.makeSound();    // Calls the overridden method in Dog class
    }
}
```

Summary

- **Compile-Time Polymorphism**:
  - Also known as method overloading.
  - Achieved by defining multiple methods with the same name but different parameter lists within the same class.
  - Resolved during compilation.
  - Static polymorphism.
- **Runtime Polymorphism**:
  - Also known as method overriding.
  - Achieved by redefining a method in a subclass with the same name and parameter list as in the superclass.
  - Resolved during runtime.
  - Dynamic polymorphism.

Both types of polymorphism are essential for creating flexible and reusable code in object-oriented programming. Compile-time polymorphism provides the ability to define multiple methods with the same name but different parameters, while runtime polymorphism allows for method implementations to be determined at runtime based on the object's type, facilitating dynamic method dispatch.

Q15. Differentiate between checked Exceptions and unchecked exceptions. Give 5 examples of checked exceptions as well as unchecked exceptions.

Checked Exceptions vs. Unchecked Exceptions

Feature	Checked Exceptions	Unchecked Exceptions
Definition	Exceptions that are checked at compile time. The compiler requires that these exceptions be either caught or declared in the method signature.	Exceptions that are not checked at compile time. They are checked at runtime and do not need to be explicitly handled or declared.
Handling	Must be handled using try-catch blocks or declared in the method signature using the throws keyword.	Can be handled optionally; not required to be caught or declared. They often represent programming errors.
Hierarchy	Subclasses of Exception (excluding RuntimeException).	Subclasses of RuntimeException.
Typical Causes	Usually represent conditions that a program should anticipate and recover from (e.g., file operations, network issues).	Typically represent programming errors or issues that are usually not recoverable by the program (e.g., logic errors, null references).

Examples	IOException, SQLException, ClassNotFoundException, NoSuchMethodException, FileNotFoundException	NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, IllegalArgumentException, ClassCastException

Examples of Checked Exceptions:

1. IOException: Thrown when an I/O operation fails or is interrupted.

```
import java.io.FileReader;
import java.io.IOException;
public class CheckedExceptionExample {
   public static void main(String[] args) {
      try {
         FileReader reader = new FileReader("nonexistentfile.txt");
      } catch (IOException e) {
         System.out.println("IOException caught: " + e.getMessage());
      }
   }
}
```

2.SQLException: Thrown when a database access error occurs.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class CheckedExceptionExample {
   public static void main(String[] args) {
      try {
         Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "user",
"password");
      } catch (SQLException e) {
         System.out.println("SQLException caught: " + e.getMessage());
      }
   }
}
```

3.ClassNotFoundException: Thrown when an application attempts to load a class through its name but the class is not found.

```
public class CheckedExceptionExample {
   public static void main(String[] args) {
      try {
         Class<?> clazz = Class.forName("com.example.NonExistentClass");
      } catch (ClassNotFoundException e) {
         System.out.println("ClassNotFoundException caught: " + e.getMessage());
      }
   }
}
```

4.FileNotFoundException: Thrown when attempting to open a file that does not exist.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
public class CheckedExceptionExample {
   public static void main(String[] args) {
      try {
         FileInputStream file = new FileInputStream("nonexistentfile.txt");
      } catch (FileNotFoundException e) {
         System.out.println("FileNotFoundException caught: " + e.getMessage());
      }
   }
```

```
}
```

1. **NoSuchMethodException**: Thrown when a particular method cannot be found.

```
import java.lang.reflect.Method;
import java.lang.NoSuchMethodException;
public class CheckedExceptionExample {
   public static void main(String[] args) {
      try {
         Method method = String.class.getMethod("nonExistentMethod");
      } catch (NoSuchMethodException e) {
         System.out.println("NoSuchMethodException caught: " + e.getMessage());
      }
   }
}
```

Examples of Unchecked Exceptions:

1.NullPointerException: Thrown when the JVM attempts to use a null object reference where an object is required.

```
java
Copy code
public class UncheckedExceptionExample {
   public static void main(String[] args) {
      String str = null;
      try {
         System.out.println(str.length());
      } catch (NullPointerException e) {
         System.out.println("NullPointerException caught: " + e.getMessage());
      }
   }
}
```

2.ArrayIndexOutOfBoundsException: Thrown when an array is accessed with an illegal index.

```
java
Copy code
public class UncheckedExceptionExample {
   public static void main(String[] args) {
      int[] array = new int[5];
      try {
         array[10] = 1;
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("ArrayIndexOutOfBoundsException caught: " + e.getMessage());
      }
   }
}
```

3.ArithmeticException: Thrown when an exceptional arithmetic condition occurs, such as division by zero.

```
java
Copy code
public class UncheckedExceptionExample {
   public static void main(String[] args) {
      try {
         int result = 10 / 0;
      } catch (ArithmeticException e) {
         System.out.println("ArithmeticException caught: " + e.getMessage());
      }
   }
```

```
}
```
4.IllegalArgumentException: Thrown when a method receives an argument that is not valid.
java
Copy code
```
public class UncheckedExceptionExample {
    public static void main(String[] args) {
        try {
            Integer.parseInt("abc");
        } catch (IllegalArgumentException e) {
            System.out.println("IllegalArgumentException caught: " + e.getMessage());
        }
    }
}
```
5.ClassCastException: Thrown when an object is cast to a type that is not a subclass of the actual type.
java
Copy code
```
public class UncheckedExceptionExample {
    public static void main(String[] args) {
        Object str = "Hello";
        try {
            Integer num = (Integer) str;
        } catch (ClassCastException e) {
            System.out.println("ClassCastException caught: " + e.getMessage());
        }
    }
}
```
Summary

- **Checked Exceptions**:
    - Are checked at compile time.
    - Must be handled by the program using try-catch blocks or declared using throws.
    - Examples: IOException, SQLException, ClassNotFoundException, FileNotFoundException, NoSuchMethodException.
- **Unchecked Exceptions**:
    - Are checked at runtime.
    - Do not require explicit handling or declaration.
    - Examples: NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, IllegalArgumentException, ClassCastException.